B2M37KASA: Komprese signálů a obrazů

Bezeztrátové kompresní algoritmy I.

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení technické v Praze

Obsah přednášky

Co už známe z poslední přednášky?

- Komprese, klasifikace a hodnocení účinnosti
- Modelování a kódování
- Informace, vlastní informace
- Pojem entropie a modely odhadu entropie zdroje
- Jednoznačně dekódovatelné kódy
- Prefixové kódy a možnost jejich reprezentace binárním stromem
- Kraft-McMillanova nerovnost – podmínka pro délku slov jednoznačně dekódovatelného kódu
- Princip minimální popisnou délku (Kolgomorovská složitost)

Typy kódování

Jednoznačně dekódovatelné

- zprávu lze dekódovat tak, že najdeme nejmenší počet znaků, který tvoří kódové slovo, tyto znaky umažeme a pokračujeme v dekódování

Prefixové

- žádné kódové slovo α_i není předponou (prefixem) jiného α_j

Optimální prefixové

- znaky vyskytující se s vyšší pravděpodobností mají kódová slova kratší
- dvě nejdelší kódová slova se liší pouze v posledním bitu a odpovídají dvěma nejřidšeji se vyskytujícím symbolům

A, B, C, D, E, F, G → 00, 10, 011, 010, 110, 1110, 1111

Blokové kódování

- kódování je blokové, jestliže všechna kódová slova mají stejnou délku
- blokové kódování je prefixové a tedy jednoznačně dekódovatelné

Příklad ASCII kód

- abeceda o 52 znacích (26 malých a 26 velkých)
- čísla (0-9), interpunkce
- všechny binární kódy stejně dlouhé (7 bitů)

  ’a’ $\longmapsto$ 01100001
  ’A’ $\longmapsto$ 01000001

Nejkratší kód

$z_1, \ldots z_r$ znaky zdrojové abecedy
$p_1, \ldots p_r$ pravděpodobnosti výskytu znaky ve zprávě

$$p_1 + p_2 + \ldots p_r = \sum_{i=1}^{r} = 1, \quad p \in \langle 0, 1 \rangle$$

$n_1, \ldots n_r$ délky kódových slov
$\overline{n}$ průměrná délka kódového slova $\overline{l}(C(Z))$

$$\overline{n} = n_1 p_1 + n_2 p_2 + \ldots n_r p_r = \sum_{i=1}^{r} n_r p_r$$

Nejkratší kód

- nejkratším s-znakovým kódováním zdrojové abecedy Z s pravděpodobnostmi výskytu P se rozumí prefixové kódování této abecedy pomocí s znaků, které má nejmenší možnou průměrnou délku slova $\overline{n}$.

Optimální kód – Shannon noisless coding theorem

Pro optimální jednoznačně dekódovatelný kód $C(A)$ ze zdrojové abecedy A do kódové abecedy B platí

$$\frac{H(A)}{log_2 m} \leq \bar{n} < \frac{H(A)}{log_2 m} + 1$$

- $H(A)$ je entropie zdroje symbolů A, m je velikost B
- redundance: $\overline{n} - H(A)/log_2 m$, také v % z $H(A)/log_2 m$
- absolutně optimální kód: $\overline{n} = H(A)/log_2 m$
- změnou zdrojové abecedy na k-tice (nezávislých) symbolů z původní abecedy $\mathcal{A}$ (rozšíření zdrojové abecedy, source extension) se lze s $\overline{n}$ k $H(A) log_2 m$ libovolně přiblížit (až do $k \to \infty$):

$$\frac{H(A)}{log_2 m} \leq \bar{n} < \frac{H(A)}{log_2 m} + \frac{1}{k}$$

Modelování

definice symbolů zpráv, resp. zdrojových komponent zpráv

- definice úzce souvisí s typem komprimovaných dat, spíš umění nez technologie

přirazení pravděpodobností výskytu hodnotám zdrojových symbolů

- statistické metody komprese

vyčleňování opakovaně se vyskytujících vzorků symbolů

- slovníkove metody komprese
- hledání korelací mezi vzory

modelování zdroje a syntéza dat

- model konečných stavů, generátor zpráv je konečný automat
- model na bázi gramatiky

Příklad

$x_1, x_2, \ldots, x_{12} = 2, 2, 4, 6, 7, 7, 7, 10, 10, 11, 11, 14$
$\mathbb{A} = \{2, 4, 6, 7, 10, 11, 14\}$

1. číslo ve dvojkové soustavě

$\Rightarrow$ 4b/znak = 48b

Příklad

$x_1, x_2, \ldots, x_{12} = 2, 2, 4, 6, 7, 7, 7, 10, 10, 11, 11, 14$
$\mathbb{A} = \{2, 4, 6, 7, 10, 11, 14\}$

2. 7 různých čísel ve dvojkové soustavě

$\Rightarrow$ 3b/znak = 36b

Příklad

$x_1, x_2, \ldots, x_{12} = 2, 2, 4, 6, 7, 7, 7, 10, 10, 11, 11, 14$
$\mathbb{A} = \{2, 4, 6, 7, 10, 11, 14\}$

3. častější číslo kratší kód

7 → `01`

11 → `111`

10 → `110`

2 → `101`

14 → `100`

4 → `000`

6 → `001`

$\Rightarrow$ 33b = 2.75b/znak

Příklad

$x_1, x_2, \ldots, x_{12} = 2, 2, 4, 6, 7, 7, 7, 10, 10, 11, 11, 14$
$\mathbb{A} = \{2, 4, 6, 7, 10, 11, 14\}$

4. kódování opakování čísla

žádné opakování → `0`
jedno opakování → `10`
dvě opakování → `11`

$7 \times 3b + 11b$

$\Rightarrow$ 32b = 2.66b/znak

Příklad

$x_1, x_2, \ldots, x_{12} = 2, 2, 4, 6, 7, 7, 7, 10, 10, 11, 11, 14$
$\mathbb{A} = \{2, 4, 6, 7, 10, 11, 14\}$

5. malé rozdíly mezi sousedními čísly

predikce

$d_1 = x_1 = 2$
$d_i = x_i - x_{i-1} = 0, 2, 2, 1, 0, 0, 3, 0, 1, 0, 3$

4b + 2b/rozdíl

$\Rightarrow$ 26b = 2.16b/znak

Příklad

$x_1, x_2, \ldots, x_{12} = 2, 2, 4, 6, 7, 7, 7, 10, 10, 11, 11, 14$
$\mathbb{A} = \{2, 4, 6, 7, 10, 11, 14\}$

6. vztah mezi čísly

predikce

$\hat{x} = i + 1$
$d_i = x_i - \hat{x_1} = 0, -1, 0, 1, 1, 0, -1, 1, 0, 0, -1, 1$

0 → `0`

-1 → `10`

1 → `11`

⇒ 19b = 1.583b/znak

Způsoby kódování

Rovnoměrné kódování

- každému znaku je přiřazen stejně dlouhý kód,
- jednodušší, rychlejší na zpracování, ale méně hospodárné,
- přiřazuje každému znaku abecedy stejně dlouhý kód bez ohledu na četnost jeho výskytu,
- typickým představitelem je Baudotovo kódování.

Nerovnoměrné kódování

- každému znaku je přiřazen jinak dlouhý kód,
- na konstrukci a zpracování je obtížnější, může však být maximálně hospodárné,
- představiteli jsou Shannon-Fanovo nebo Huffmanovo kódování.

Přístup k informaci

Sémantický (logické algoritmy)

- studuje sémantiku (logickou informační hodnotu) zdrojových dat
- zpravidla se jedná o ztrátovou kompresi

Syntaktický (fyzické algoritmy)

- nezáleží na sémantice dat a pracuje bez zřetele na logiku dat
- nové sekvence symbolů (bitů, bajtů, ...), pokud možno kratší

Pragmatický

- studuje se účinek zdrojových dat na příjemce kódované zprávy

Statické a adaptivní algoritmy

Statická (neadaptivní komprese)

- Neměnná procedura komprese, neměnný model
- Procedura komprese nezávislá na vlastnostech aktuálně komprimovaných dat

Adaptivní komprese

- Dynamicky se měnící procedura komprese, proměnný (adaptivní) model
- Procedura komprese respektující statistické vlastnosti konkrétních aktuálně komprimovaných dat

Většina algoritmů má obě varianty.

Míry kompresních algoritmů

Časová a paměťová složitost / náročnost

- asymptotická složitost
- experimentální náročnost – test na referenčních datech

Míry komprese

- kompresní poměr (compression ratio) = poměr velikosti originálních a komprimovaných dat, také jako procento velikosti komprimovaných dat z velikosti originálních dat
- compression rate = průměrná velikost komprimovaných dat na vzorek originálních dat, např. pixel u obrazu – bitů/pixel

Míra zkreslení

- rozdíl mezi originálními a dekomprimovanými daty, více způsobů měření „přesnosti (fidelity)“ a „kvality“ obsahu

Míry komprese

- komprese
  - kódování zprávy délky $O(X)$ [b] na zprávu délky $L(X)$ [b]
  - cílem je dosažení $L(X) \ll O(X)$
  - ideálně $L(X) \to H(X)$, kde $H(X)$ je redundance
  - méně než $H(X)$ nelze kódovat bezeztrátově
- kompresní poměr, $L(X)/O(X)$
- kompresní faktor, $O(X)/L(X)$
- kompresní zisk, $(O(X) - L(X))/O(X)$
- redundance zprávy, $R(X) = O(X) - H(X)$
- redundance komprimované zprávy, $\overline{R}(X) = L(X) - H(X)$

Experimentální měření účinnosti - datové korpusy

Calgary Corpus (1987)

- `http://corpus.canterbury.ac.nz/descriptions/#calgary`
- 14 souborů – text, grafika, binární soubory

Canterbury Corpus (1997)

- `http://corpus.canterbury.ac.nz`
- 11 souborů + artifical c. (4) + large c. (3) + miscellaneous c. (1)

Silesia Corpus (2003)

- `http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia`
- 18 souborů o velikostech 6 – 51MB

Prague Corpus (2011)

- Jan Holub et al., FIT ČVUT, Praha

Bezeztrátové algoritmy

- RLE (Run-length)
- Move-To-Front (Bentley)
- Shannon-Fano
- Huffman
- Arithmetic
- Lempel-Ziv (Welsh)
- ...

Intuitivní metody

Baudotův kód

- 5 bitový kód, 62 znaků
- řídící symboly: změna registru (písmena / čísla), konec řádku, ...
- telex, dálnopis

Ad hoc komprese textu

- vynechání mezer + bitová mapa poloh mezer
- kódování ASCII kódů (7b) v 8b bytech
- kódování čistě číselných dekadických dat - 2 číslice v byte

MacWrite

WYSIWYG word processor, Apple Macintosh (1984)
Symbolem je slabika obsahující

- jeden ASCII znak
- dva komprimované znaky

Komprimované znaky

- 4-bitové kódy pro 15 nejčastěji používaných znaků
- pro zobrazení ostatních znaků se použije slabika se znakem ESCAPE následovaná v další slabice plným ASCII znakem

Každý odstavec (paragraf) se uchovává v komprimované nebo v nekomprimované verzi podle toho, které verze je kratší
Indikace komprimovaný /nekomprimovaný odstavec +1 bit

Run Length Encoding, RLE

⇒ náhrada skupin opakujících se symbolů (runs) čítačem opakování a specifikací opakovaného symbolu

Příklad

- WWWWWWWWWWWWWBWWWWWWWWWWWWWWBBBWWWWWWWW
  WWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWWWW
- 12W1B12W3B24W1B14W

Bytový proud – např. jeden byte určuje počet opakování kódovaného bytu, druhý byte udává hodnotu kódovaného bytu
Nevhodná vstupní data – může být neúčinná a velikost proudu z výstupu kodéru až dvojnásobná oproti vstupu

- až pro posloupnosti delší než určitý počet symbolu, např. 3

Aplikace – text, obraz (BMP)

RLE, MNP class 5

RLE komprese používaná historicky v modemech MNP, Microcom, Inc., Microcom Network Protocol (in flight coding)
Dosahoval až 200% účinnost
Základem je metoda RLE

- 3 identické byty vyslané bezprostředně po sobě spouští RLE
- za tyto 3 byty se vsouvá repetiční indikátor (byte)
- pokud se v opakování nepokračuje, repetiční indikátor je 0
  - opakování 3 bytů způsobuje expanzi zprávy AAA → AAA0
  - opakování 4 bytů zachová původní délku zprávy AAAA → AAA1
  - ke kompresi dojde při opakování 5 a více bytů AAAAAA → AAA2
- počet repetic je uměle omezen na hodnotu 250

Pro kódování násobně použitých znaků v textu se používá dynamicky udržovaná tabulka bitově kratších kódových slov

ITU-T T4 (Group 3) - standard FAXových strojů

Princip

- Každý řádek stránky je tvořený střídáním posloupností bílých (w) a černých (b) pixelů
- předpokládá se, ze řádek začíná bílým pixelem
- kódují se pouze délky těchto posloupností
- bbbbwwbbbbb → 1,4,2,5

Implementace

- Huffmanovo kódovaní s přednastavenou kódovou knihou
- odlišné kódování pro délky sledů černých a bílých pixelů
  - pro vyjádření délek posloupností pixelů větších než 64 se používají definovaná kódová slova pro násobky 64
  - délka posloupnosti 150 pixelů se kóduje kódovým slovem pro délku 128 pixelů + kódovým slovem pro délku 22 pixelů

Move-to-Front kódování

- dynamicky se vytváří a udržuje abeceda $\mathbb{A}$
- kód znaku odpovídá jeho pořadí v $\mathbb{A}$
  - je dané dosavadní frekvencí výskytu znaku v kódované zprávě
  - kód prvku v $\mathbb{A}$ platný pouze v jistém okamžiku

$\mathbb{A}$ = thes, e $\rightarrow$ 2
$\mathbb{A}$ = eths

- menším indexům (celým číslům) se přidělují kratší kódová slova než větším indexům
- uplatňuje se tzv. lokální adaptivita, efektivní v případech, kdy se koncentruje v jistém místě v souboru identické symboly

Move-to-Front kódování

Strategie přidělení indexů

Nechť $\mathbb{A}$ je 8 prvková abeceda $\mathbb{A} = \{a, b, c, d, m, n, o, p\}$

- Indexy lze označit $\langle 1, \ldots, 8 \rangle$
- Binárně: 1, 10, 11, ..., 1000
- Požadavek na bezprostřední jednoznačné dekódování
  - Blokový kód – neefektivní
  - Kódová slova proměnné délky – prefixový kód
- Každá hodnota indexu i se prefixuje $log_2 i$ počtem nul
  - 1, 010, 011, 0011, ..., 0001000

Dochází ke změně entropie textu, používá se jako předkrok k Huffmanově kódování

Move-to-Front kódování

Příklad kódování řetězce

abcap

- a → 1 — A = a, b, c, d, m, n, o, p
- ab → 1 010 — A = b, a, c, d, m, n, o, p
- abc → 1 010 011 — A = c, b, a, d, m, n, o, p
- abca → 1 010 011 011 — A = a, c, b, d, m, n, o, p
- abcap → 1 010 011 011 0001000 — A = p, a, c, b, d, m, n, p

Kódování čísel

- kódování přirozených čísel – celá lze bijektivně zobrazit na přirozená
- předpokládáme, že větší čísla se vyskytují s menší pravděpodobností
- binární kódy s proměnnou délkou
  - proměnná délka kódových slov pro zdrojová slova pevné délky
  - nízká průměrná délka kódu vs. náročnější manipulace s kódem

V porovnání s blokovým kódem, s využitím bufferu

Unární kód

- kódování přirozených čísel
- **Alpha** – $i \geq 0$ zřetězení `1` a `0` (nebo opačně)
- prefixový, délka $i + 1$, optimální při $P(i) = 2^{-i}$
- $\alpha(1) = 1$, $\alpha(i + 1) = 0\alpha(i)$

Binární kód

- **Beta** – $\beta(0) = 0$, $\beta(1) = 1$, $\beta(2i) = \beta(i)0$, $\beta(2i+11) = \beta(i)1$
- vlastnosti: $|\beta(i)| = \lfloor log(i) \rfloor + 1$, není prefixový
- binární kód s **pevnou délkou**
  - $|A| = n$ – pevná délka kódového slova
  - optimální kód pro rozdělení pravděpodobnosti $p(i) = \frac{1}{n}$
- binární kód s **rostoucí délkou**
  - n je aktuální velikost slovníku
  - slovník roste → délka kódu se prodlužuje
- co když n není mocnina 2?

Minimální binární kód β_n

- $i < 2^k - n - (k-1)$ bitový binární kód i
- jinak – k bitový binární kód pro $i + 2^k - n$

číslo	kód
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	1110
8	1101

Eliasovy kódy

- Peter Elias (1975)

- **Redukovaný binární kód** $\beta'(i) = \beta(i)$ bez prvního bitu
- $\gamma'(i) = \alpha(|\beta(i)|)\beta'(i)$
  - $|\gamma'(i)| = 2\lfloor log(i) \rfloor + 1$
- $\gamma(i)$ je permutace $\gamma'(i)$
  - každý bit unárního kódu α následován bitem binárního kódu β'
- $\delta(i) = \gamma(|\beta(i)|)\beta'(i)$
- v konstrukci lze analogicky pokračovat
- Vlastnosti:
  - γ, γ' jsou dlouhé pro velká i
  - δ je dlouhé pro kratší i
  - v praxi postačuje δ – $|\gamma(10^9)| = 59$, $|\delta(10^9)| = 39$, $|\varepsilon(10^9)| = 39$

Golombův kód

- Solomon Wolf Golomb (1966)

- Golombův kód $G_b(i)$, $i = 1, 2, 3, \ldots$ s parametrem $b \in 1, 2, 3, \ldots$
- Kombinace unárního a minimálního binárního kódu
- G_b – selektor skupiny | pozice ve skupině:

$$G_b(i) = \alpha\left(\lfloor (i-1)/b \rfloor + 1\right) \beta_b\left((i-1) \bmod b\right)$$

i	1	2	3	4	5	6	7	8	9
$b = 1$	1	01	001	0001	00001	000001	0000001	00000001	000000001
$b = 5$	100	101	110	1110	1111	0100	0101	0110	01110

- optimální prefixový kód pro geometrické rozdělení $p(i) = (1-p)^{i-1}p$

Riceův kód

- Robert F. Rice (1979)

- Riceův kód $R_k(i) = G_{2^k}(i)$
- kombinace unárního a binárního kódu: $R_k = \alpha(xxxxxxxx + 1)|$dolních k bitů

i	1	2	3	4	5	6	7	8	9
$k = 1$	10	11	010	011	0010	0011	00010	00011	000010
$k = 5$	100	101	110	111	0100	0101	0110	0111	00100

Entropie zdroje

Abeceda $\mathbb{A} = \{A, B, C, D\}$
Pravděpodobnosi výskytu $\frac{3}{4}$, $\frac{1}{8}$, $\frac{1}{16}$, $\frac{1}{16}$

Entropie zdroje

$-\frac{3}{4} log_2\left(\frac{3}{4}\right) - \frac{1}{8} log_2\left(\frac{1}{8}\right) - \frac{1}{16} log_2\left(\frac{3}{16}\right) - \frac{1}{16} log_2\left(\frac{1}{16}\right) = 1.186\,[b/symbol]$

Kódování 1

A = 00, B = 01, C = 10, D = 11, průměr 2 [b/symbol]

Kódování 2 (prefixový kód, slova proměnné délky)

A = 0, B = 10, C = 110, D = 111, průměr 1,375 [b/symbol]

Kódování 2 dosahuje průměrné délky symbolu bližší entropii než kódování 1 → je efektivnější.

Tunstallův kód

⇒ zdrojová slova proměnné délky kódována na kódová slova pevné délky $k \geq log_m n$ kódových symbolů ⇒ blokový kód, ∼ variable-to-fixed code (n je velikost zdrojové abecedy, m je velikost kódové abecedy)

Základní myšlenka

- Mějme abecedu $\mathbb{X} = \{x_1, \ldots, x_i, \ldots, x_m\}$
- Nechť x_i je prvek abecedy s nejvyšší pravděpodobností výskytu
- Vyřaďmě x_i z abecedy $\mathbb{X}$ a doplňme ji dvoupísmennými řetězci $x_i x_1, \ldots, x_i x_m$
- Pokud $|\mathbb{X}| + (m-1) < 2^n$, postup opakujme
- Výsledkem je reprezentace prvků abecedy n-bitovými čísly

Shannon-Fanova metoda

Statistická metoda bezeztrátové komprese navržená roku 1949

- Claude Shannon[1]
- Robert Fano
- Weaver

Od Huffmanova kódování se liší pouze konstrukcí binárního stromu

- Konstruován o kořene směrem k listům
- Rekurzivní algoritmus

Řešení nemusí být optimální, proto se téměř nepoužívá.

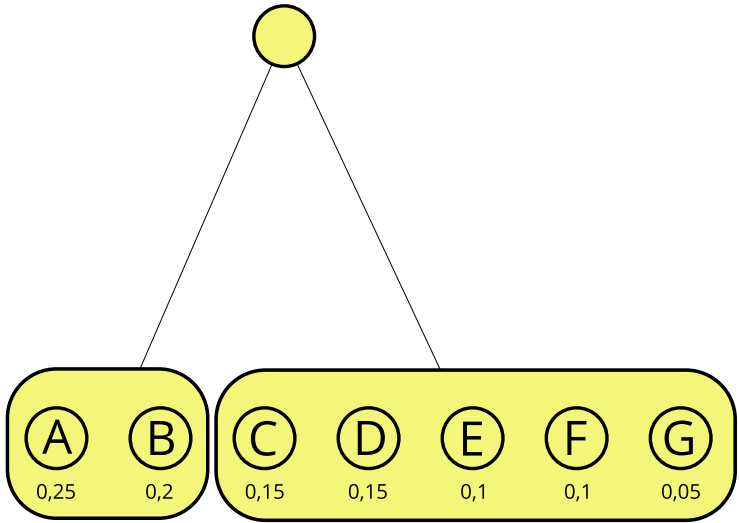
Shannon-Fanova metoda

Konstrukce binárního stromu

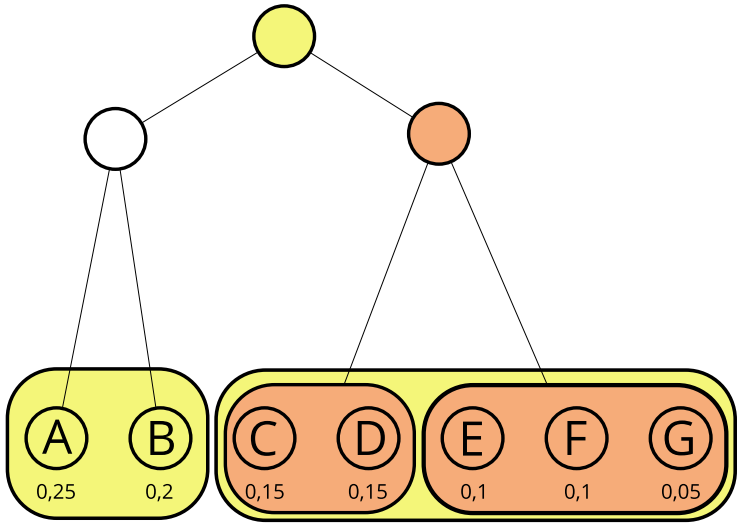
1. Zdrojové znaky uspořádáme podle pravěpodobností p_i
2. Zdrojové znaky se rozdělí na dvě podskupiny tak, aby součet pravděpodobností byl v obou skupinách přibližně stejný
3. První skupině se přiřadí symbol 0, druhé 1
4. Poslední dva kroky opakujeme, dokud v každé podskupině nezůstane jen jeden znak

Shannon-Fanova metoda, příklad

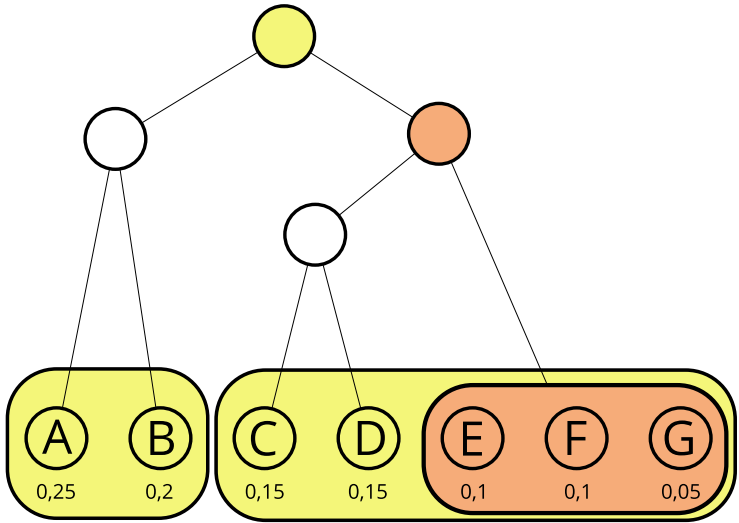
Shannon-Fanova metoda, příklad



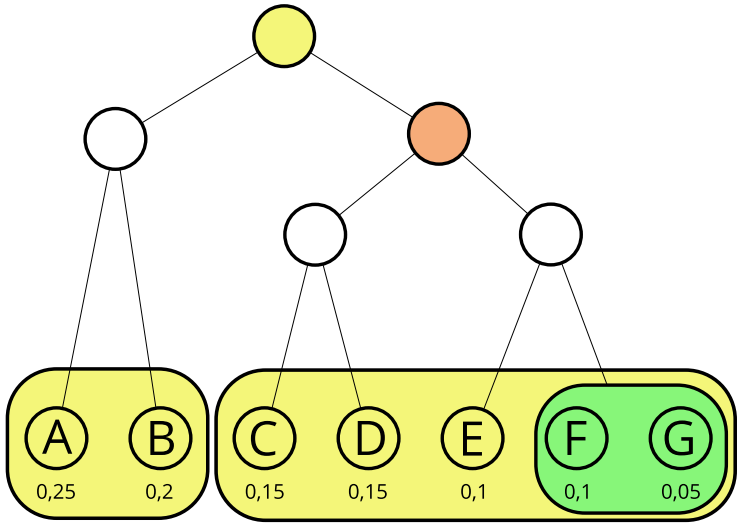
Shannon-Fanova metoda, příklad



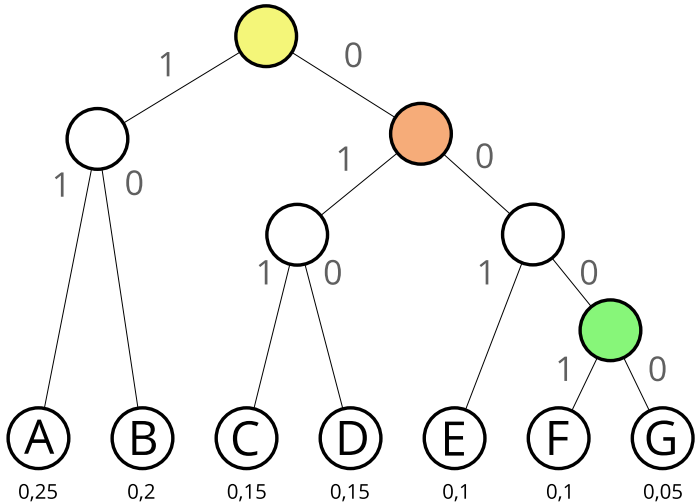
Shannon-Fanova metoda, příklad



Shannon-Fanova metoda, příklad



Shannon-Fanova metoda, příklad



Huffmanovo kódování

David Albert Huffman (1925 – 1999)

- Vyvinul kódování jako doktorand na MIT v roce 1952[2]
- Lepší kód než metoda jeho školitele (Claude Shannon :-)

Významné využití v kompresi obrazu, zvuku a textu, ...

- Návrh kódu pro známý pravděpodobnostní model zdroje
- Návrh kódu pro neznámé statistiky zdroje

Huffmanův kód

- Prefixový kód
- optimální pro daný model (množinu pravděpodobností)

Postup generování Huffmanova kódu

Založen na dvou vlastnostech optimálních prefixových kódů

- v optimálním kódu mají častěji se vyskytující symboly (větší pravděpodobnost výskytu) kratší kódová slova
- v optimálním kódu, dva symboly vyskytující se nejméně často, mají stejnou délku kódového slova

Postup lze získat přidáním jednoho požadavku

- kódová slova pro dva nejméně pravděpodobné symboly liší pouze v posledním bitu
  - pro dva nejméně pravděpodobné symboly γ a δ, pokud $m * 0$ je kódové slovo pro γ, $m * 1$ je kódové slovo pro δ
  - symbol $*$ značí sloučení a m je binární číslo

Huffmanovo kódování – příklad postupu

$\mathbb{A} = \{a_1, a_2, a_3, a_4, a_5\}$, $P = (0.2, 0.4, 0.2, 0.1, 0.1)$, $H = 2.122\, b/sym$

Seřazení symbolů podle pravděpodobnosti

Znak	Pravděpodobnost	Kódové slovo
a_2	0,4	$c(a_2)$
a_1	0,2	$c(a_1)$
a_3	0,2	$c(a_3)$
a_4	0,1	$c(a_4)$
a_5	0,1	$c(a_5)$

Dva nejméně pravděpodobné symboly

- $c(a_4) = \alpha_1 * 0$
- $c(a_5) = \alpha_1 * 1$

kde α_1 je řetězec binárních znaků.

Huffmanovo kódování – příklad postupu

$\mathbb{A}' = \{a_1, a_2, a_3, a'_4\}$, $P(a'_4) = P(a_4) + P(a_5) = 0,2$

Znak	Pravděpodobnost	Kódové slovo
a_2	0,4	$c(a_2)$
a_1	0,2	$c(a_1)$
a_3	0,2	$c(a_3)$
a'_4	0,2	α_1

Dva nejméně pravděpodobné symboly

- $c(a_3) = \alpha_2 * 0$
- $c(a'_4) = \alpha_2 * 1 = \alpha_1$

Zřejmě

- $c(a_4) = \alpha_2 * 10$
- $c(a_5) = \alpha_2 * 11$

Huffmanovo kódování – příklad postupu

$\mathbb{A}'' = \{a_1, a_2, a'_3\}$, $P(a'_3) = P(a_3) + P(a'_4) = 0,4$

Znak	Pravděpodobnost	Kódové slovo
a_2	0,4	$c(a_2)$
a'_3	0,4	α_2
a_1	0,2	$c(a_1)$

Dva nejméně pravděpodobné symboly

- $c(a'_3) = \alpha_3 * 0 = \alpha_2$
- $c(a_1) = \alpha_3 * 1$

Zřejmě

- $c(a_3) = \alpha_3 * 00$
- $c(a_4) = \alpha_3 * 010$
- $c(a_5) = \alpha_3 * 011$

Huffmanovo kódování – příklad postupu

$\mathbb{A}''' = \{a_3'', a_2'\}$, $P(a_3'') = P(a_3') + P(a_1) = 0,6$

Znak	Pravděpodobnost	Kódové slovo
a_3''	0,6	α_3
a_2	0,4	$c(a_2)$

Zbývají pouze dva znaky

- $c(a_3'') = 0 = \alpha_3$
- $c(a_2) = 1$

Zřejmě

- $c(a_1) = 01$
- $c(a_3) = 000$
- $c(a_4) = 0010$
- $c(a_5) = 0011$

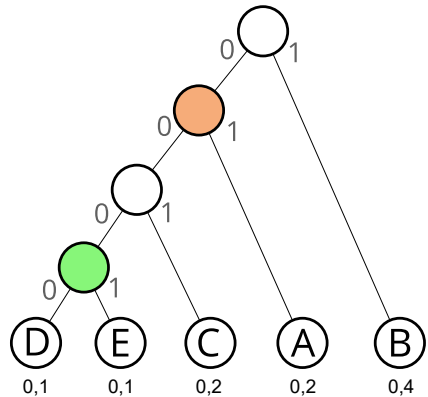
Alternativní Huffmanovy kódy

Pokud se ve vstupní abecedě vyskytují symboly se stejnou pravděpodobností výskytu, lze binární strom budovat ve více alternativách.

Kód 1

- tzv. kanonický kód

symbol	P	kód
A	0,2	01
B	0,4	1
C	0,2	001
D	0,1	0000
E	0,1	0001



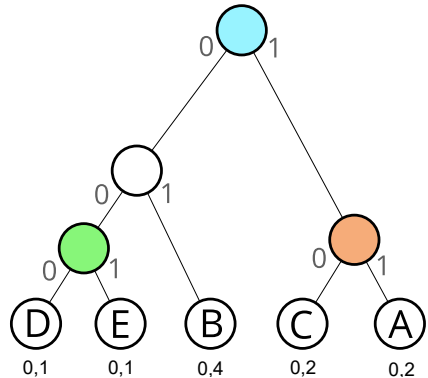
Alternativní Huffmanovy kódy

I následující kód je optimální. Na rozdíl od kanonické formy vykazuje menší odlišnost kódových slov, což může být v některých aplikacích výhodnější.

Kód 2

- redukce délky vyrovnávací paměti
- konstatní rychlost přenosu

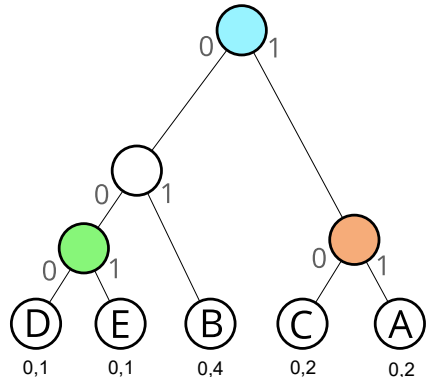
symbol	P	kód
A	0,2	11
B	0,4	01
C	0,2	10
D	0,1	000
E	0,1	001



Alternativní Huffmanovy kódy

Kód 3

symbol	P	kód
A	0,2	011
B	0,4	1
C	0,2	010
D	0,1	000
E	0,1	001



Huffmanovo kódování - shrnutí

- Kód má minimální průměrnou délku kódového slova
  - Huffmanův kód je pro daný model textu optimální
  - Humanovo kódován maximálně potlačuje redundanci
- Pro jistou třídu textů lze vytvořit kódovou knihu
  - ušetří se doba věnovaná generování stromu
  - kód ale nemusí být optimální (používá se kódová kniha třídy textů, nikoli kódová kniha daného textu)
- Originálním zprávám shodné délky nemusí odpovídat komprimované zprávy shodné délky
  - to může být někdy implementační problém

Aritmetické kódování

Motivace

- pro malé abecedy může být i Huffmanovo kódování neefektivní díky redundancím v kódových slovech symbolů, pokud nejsou pravděpodobnosti výskytů jednotlivých znaků mocninami dvou
- řešením by mohlo být shlukování m symbolů do bloků – slov, a generování kódových slov pro všechny posloupnosti znaků délky m
- složitost by ale rostla exponenciálně – chybné řešení

Arimetické kódování

- technika umožňující informace ze symbolů tvořících zprávu kombinovat tak, že sdílí stejné bity
- hodnota výsledneho kódového slova se asymptoticky blíží součtu množství informací obsažených v jednotlivých symbolech

Aritmetické kódování

Motivace

- pro malé abecedy může být i Huffmanovo kódování neefektivní díky redundancím v kódových slovech symbolů, pokud nejsou pravděpodobnosti výskytů jednotlivých znaků mocninami dvou
- řešením by mohlo být shlukování m symbolů do bloků – slov, a generování kódových slov pro všechny posloupnosti znaků délky m
- složitost by ale rostla exponenciálně – chybné řešení

Arimetické kódování

- technika umožňující informace ze symbolů tvořících zprávu kombinovat tak, že sdílí stejné bity
- hodnota výsledneho kódového slova se asymptoticky blíží součtu množství informací obsažených v jednotlivých symbolech

Efektivita aritmetického kódování

Mějme zprávy o 1000 symbolech, pravděpodobnost výskytu jedné konkrétní hodnoty znaku $p = 0,999$

- množství informace v jednom symbolu log2 1/0,999 = 0,00144 b
- množství informace v 1000 znakových zprávách je 1,4 b

Huffmanovo kódování

- kódové slovo délku alespoň 1b.
- délka komprimované zprávy bude alespoň 1000 bitů

Aritmetické kódování

- při tomto rozložení pravděpodobností může generovat jedinné kódové slovo o délce např. 3 bity
  - vhodné tam, kde se v distribuční funkci vyskytují velké pravděpodobnosti
  - Pokud jsou pravděpodobnosti malé, přednosti AK nejsou významné

Efektivita aritmetického kódování

Mějme zprávy o 1000 symbolech, pravděpodobnost výskytu jedné konkrétní hodnoty znaku $p = 0,999$

- množství informace v jednom symbolu log2 1/0,999 = 0,00144 b
- množství informace v 1000 znakových zprávách je 1,4 b

Huffmanovo kódování

- kódové slovo délku alespoň 1b.
- délka komprimované zprávy bude alespoň 1000 bitů

Aritmetické kódování

- při tomto rozložení pravděpodobností může generovat jedinné kódové slovo o délce např. 3 bity
  - vhodné tam, kde se v distribuční funkci vyskytují velké pravděpodobnosti
  - Pokud jsou pravděpodobnosti malé, přednosti AK nejsou významné

Efektivita aritmetického kódování

Mějme zprávy o 1000 symbolech, pravděpodobnost výskytu jedné konkrétní hodnoty znaku $p = 0,999$

- množství informace v jednom symbolu log2 1/0,999 = 0,00144 b
- množství informace v 1000 znakových zprávách je 1,4 b

Huffmanovo kódování

- kódové slovo délku alespoň 1b.
- délka komprimované zprávy bude alespoň 1000 bitů

Aritmetické kódování

- při tomto rozložení pravděpodobností může generovat jedinné kódové slovo o délce např. 3 bity
  - vhodné tam, kde se v distribuční funkci vyskytují velké pravděpodobnosti
  - Pokud jsou pravděpodobnosti malé, přednosti AK nejsou významné

Princip

Východiska

- každá posloupnost n symbolů se reprezentuje jedinečným intervalem na číselné ose v intervalu < 0, 1), takových intervalů může být nekonečně mnoho
- vstupní posloupnosti symbolů nabývajících hodnot n-znakové abecedy s pravděpodobnostmi výskytu znaku $p_1, \ldots, p_n$

Algoritmus

- přidělí interval odpovídající kumulativní distribuční funkci
- začíná intervalem o rozměru 1 (tj. 0 až 1)
- zužuje interval faktorem pi pro každý další znak i v posloupnosti

Princip

Východiska

- každá posloupnost n symbolů se reprezentuje jedinečným intervalem na číselné ose v intervalu < 0, 1), takových intervalů může být nekonečně mnoho
- vstupní posloupnosti symbolů nabývajících hodnot n-znakové abecedy s pravděpodobnostmi výskytu znaku $p_1, \ldots, p_n$

Algoritmus

- přidělí interval odpovídající kumulativní distribuční funkci
- začíná intervalem o rozměru 1 (tj. 0 až 1)
- zužuje interval faktorem pi pro každý další znak i v posloupnosti

Princip

Východiska

- každá posloupnost n symbolů se reprezentuje jedinečným intervalem na číselné ose v intervalu < 0, 1), takových intervalů může být nekonečně mnoho
- vstupní posloupnosti symbolů nabývajících hodnot n-znakové abecedy s pravděpodobnostmi výskytu znaku $p_1, \ldots, p_n$

Algoritmus

- přidělí interval odpovídající kumulativní distribuční funkci
- začíná intervalem o rozměru 1 (tj. 0 až 1)
- zužuje interval faktorem pi pro každý další znak i v posloupnosti

Bity nutné pro jedinečnou identikaci intervalu o rozměru *s* můžeme chápat jako entitu a dát do spojitosti délku její reprezentace s množstvím informace v symbolech zprávy.

Příklad

Zadání

Nechť zprávy jsou tvořeny 2 znaky a symbolem konce zprávy (N = 3), odpovídající distribuční funkce jsou:

- pdf: p1 = 0,6; p2 = 0,3; p3 = 0,1
- cdf (cumulative): F(1) = 0,6; F(2) = 0,9; F(3) = 1

Postup

- Interval 0 až 1 se rozdělí na N intervalů
- šířky intervalů jsou úměrné pravděpodobnostem výskytů prvků abecedy, tj. hranice intervalu vymezuje cdf
- zpráva začínající i-tým prvkem vstupní abecedy je reprezentovaná číslem z intervalu vymezenými i-tému prvku vstupní abecedy
- pro druhý a další symbol zprávy se postup rekurzivně opakuje vždy v rámci intervalu vymezeného předchozím znakem zprávy

Příklad

Zadání

Nechť zprávy jsou tvořeny 2 znaky a symbolem konce zprávy (N = 3), odpovídající distribuční funkce jsou:

- pdf: p1 = 0,6; p2 = 0,3; p3 = 0,1
- cdf (cumulative): F(1) = 0,6; F(2) = 0,9; F(3) = 1

Postup

- Interval 0 až 1 se rozdělí na N intervalů
- šířky intervalů jsou úměrné pravděpodobnostem výskytů prvků abecedy, tj. hranice intervalu vymezuje cdf
- zpráva začínající i-tým prvkem vstupní abecedy je reprezentovaná číslem z intervalu vymezenými i-tému prvku vstupní abecedy
- pro druhý a další symbol zprávy se postup rekurzivně opakuje vždy v rámci intervalu vymezeného předchozím znakem zprávy

Dekódování

Postup inverzní ke kódování:

- komprimovaná zpráva (kódové slovo) = číslo v jistém podintervalu z intervalu 0 až 1
- podinterval určí hodnotu prvního symbolu a následně se analyzovaný interval zúží podle jeho hodnoty
- algoritmus pokračuje stejným postupem až do detekce znaku konce zprávy nebo do získání originální zprávy předem známé délky

Délka kódového slova

Jak dlouhé je (binární) kódové slovo?
Jak efektivně určit posloupnost bitů reprezentujících interval, resp. číslo v intervalu?

- Jako značku lze použít střed intervalu
- Platí (bez důkazu :-)
  - Pro posloupnost n symbolů obsahujících množství informace $s_1, s_2, \ldots, s_n$ lze délku binárního kódového slova reprezentujcho celou posloupnost shora oříznout na

$$2 + \sum_{i=1}^{N} s_i$$

  a toto kódové slovo nebude prefixem kódového slova žádné jiné posloupnosti n symbolů

Algoritmy kódování a dekódování

Algoritmus kódování

1: $d \leftarrow 0,\ r \leftarrow 1$
2: **while** read(znak) != EOF **do**
3: $\quad d \leftarrow r * low(znak)$
4: $\quad r \leftarrow r * (high(znak) - high(znak))$
5: **end while**
6: **return** $code(z),\ z \subseteq \langle d, d + r \rangle$

Algoritmus dekódování

1: $d \leftarrow 0,\ r \leftarrow 1$
2: **repeat**
3: $\quad find(znak),\ low(znak) \leq (code - d)/r < high(znak)$
4: $\quad d \leftarrow d + r * low(znak)$
5: $\quad r \leftarrow r * (high(znak) - low(znak))$
6: **until**

Příklad

Zpráva: BILL GATES

- pravděpodobnosti výskytů znaků s výjimkou $p(L) = 0{,}2$ jsou $p(i) = 0,1$
- vymezení intervalů:

znak	p	podinterval
mezera	1/10	$\langle 0.0, 0.1)$
A	1/10	$\langle 0.1, 0.2)$
B	1/10	$\langle 0.2, 0.3)$
E	1/10	$\langle 0.3, 0.4)$
G	1/10	$\langle 0.4, 0.5)$
I	1/10	$\langle 0.5, 0.6)$
L	2/10	$\langle 0.6, 0.8)$
S	1/10	$\langle 0.8, 0.9)$
T	1/10	$\langle 0.9, 1.0)$

znak	dolní mez	horní mez
	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0.256	0.258
L	0.2572	0.2576
Mezera	0.2572	0.25724
G	0.257216	0.257220
A	0.2572164	0.2572168
T	0.25721676	0.2572168
E	0.257216772	0.257216776
S	0.2572167752	0.2572167756

Výsledek

- značkou zprávy BILL GATES je např. 0.2572167754
- převod na binární zobrazení
- Plná délka:

  0.01000001110110001111010101100110010100110101100100001
- Kódové slovo

  01000001110110001111010101100 11
- délka BILL GATES v ASCII = 70 bitů
- délka BILL GATES aritmeticky komprimovaný = 31 bitů
- tato délka je shodná s délkou komprimované zprávy BILL GATES Huffmanovým kódováním

Poznámky

- Výsledkem kódování je libovolné číslo z posledního intervalu.
- Je potřeba zakódovat informaci o ukončení, protože dekodér musí vědět, kdy má s dělením intervalu skončit
- Existují dvě varianty řešení tohoto problému:
  - před vlastním kódem čísla z intervalu je nějakým prefixovým kódem zakódován počet symbolů
  - použije se speciální znak pro označení konce vstupu, zarážka. Tento znak bude mít nejnižší pravděpodobnost. Pokud na něj dekodér narazí, pozná, že dekódování je u konce
- Tak jako statická verze Huffmanova kódování musí nejprve určit pravděpodobnosti výskytů znaků, tyto frekvence předat dekodéru a teprve pak kódovat, musí toto provést i statická verze aritmetického kódování.

Příklad ukončení vstupu

Kódujme text *abacaacbc*, rozšíříme o ukončovací znak #

- Četnosti jednotlivých znaků *a*, *b*, *c* a # ve zprávě jsou 4, 2, 3, 1
- Tomu odpovídají pravděpodobnosti 0,4; 0,2; 0,3; 0,1
- Úkol – odvoďte, že
  - textu odpovídá interval $\langle 0,1803860992; 0,180388864\rangle$,
  - pravděpodobnost výskytu této zprávy je 0,000002765
  - množství informace ve této zprávě je 18,46439347 bitů
- Vhodný kandidát: 0,1803874969482421875 0,00101110001011011111B
- Kódové slovo, bez úschovy četnosti symbolů, je dlouhé 19 bitů.
- Kódové slovo textu v ASCII kódu má délku 70 bitů

Adaptivní algoritmus

Dekodér může použít předem připravený pravděpodobnostní model (např. pro jeden typ dat – knihy v češtině)
Dalším řešením tohoto problému je použití adaptivní verze

- Adaptivní verze AK na základě pravděpodobností kóduje jednotlivé znaky stejně jako statické aritmetické kódování
- ale model pravděpodobnostního rozdělení se mění po zakódování každého znaku, např.
  - množinu znaků kodér i dekodér zná předem, např, a, b, c, #
  - na počátku předpokládá četnost každého znaku 1, tj. pravděpodobnosti 1/4, 1/4, 1/4, 1/4
  - zakódování jednoho znaku (např. a) zvýší jeho četnost o 1 a pro další krok přepočte pravděpodobnosti 2/5, 1/5, 1/5, 1/5

Nevýhody aritmetické kódování

- Kódování znaků s malou pravděpodobností výskytu
  - způsobí výraznější zmenšení délky intervalu a tím výraznější zvětšení délky čísla
  - v textu se vyskytují zřídka a jejich celkový vliv na délku zakódovaného čísla je malý
- S rostoucí délkou kódovaného textu roste i délka čísel vyjadřující meze intervalu
- Pro praktické použití se
  - používají jen celá čísla, což je výhodnější, než použití desetinných čísel
  - pracuje s čísly s omezeným počtem míst
    - cenou je určité zhoršení kompresního poměru, které je ale zanedbatelné

Úvod

- Místo intervalu $I = 0, 1)$ použijeme celočíselný interval $I =< 0, M >$
- Je účelné použít rozsah hodnot, jaký poskytují běžné datové typy
- Maximální hodnota 32-bitového čísla bez znaménka je $2^{32} - 1$
- Při výpočtu ale může být k rozsahu intervalu M přičtena 1
  - u hodnoty $2^{32} - 1$ by došlo k přetečení
  - jako horní mez použitého intervalu zvolíme číslo o jeden bit menší $M = 2^{31} - 1$
- Místo pravděpodobností použijeme četnosti výskytů znaků (frekvence).
  - $f_1, f_2, ..., f_n$, kde n je počet znaků kódovatelných pomocí 1B, tj. 256 bitů
- Kumulativní četnosti

$$g_0 = 0, g_1 = f_1, g_2 = f_1 + f_2, ..., g_n = f_1 + f_2 + ... + f_n$$

$$p_1 = \frac{f_1}{g_n}, ..., p_n = \frac{f_n}{g_n}$$

Princip

1. Na počátku hodnotu intervalu I položíme $I =< 0, M >$
2. Kódování každého znaku
   - vyjdeme ze stávající hodnoty intervalu $I =< low, high >$
   - interval I rozložíme na disjunktní celočíselné intervaly podle frekvencí jednotlivý znaků:

znak	odpovídající interval
a_1	$I_1 =< low + s * g_0, low + s * g_1 - 1 >=< low, low + s * g_1 - 1 >$
a_i	$I_i =< low + s * g_{i-1}, low + s * g_i - 1 >$
a_n	$I_n =< low + s * g_{n-1}, low + s * g_n - 1 >$

kde (pozor, celočíselné dělení):

$$s = \frac{(high - low + 1)}{g_n}$$

Limity

Velikost kumulativních frekvencí znaků

- omezena rozsahem zvoleného datového typu (32 bitů)
- je zapotřebí s nimi provádět určité aritmetické operace

⇓

- hodnotu g_n omezíme na maximální rozsah 29 bitů $\rightarrow g_n \leq 2^{29} - 1$

Pokud kumulativní četnost překročí hodnotu g_n

- snížení hodnot četností

⇓

- všechny četnosti větší než 1 se dělí 2
- přepočet kumulativních četností

Zvětšení intervalu

Kódování znaku a_i znamená výpočet nové hodnoty intervalu I

- délka intervalu se při kódování neustále zmenšuje,
- interval by se po určité době stal tak malým, že výpočet by přestal být korektní
  - na rozdíl od neceločíselného aritmetického kódování

$\Downarrow$

- rozsah intervalu I je proto při kódování průběžně zvětšován
  - informace o zvětšení kódována formou příznakového bitu

Referenční hodnoty

- čtvrtina maximálního rozsahu $Q_1 = (M + 1)/4 = 2^{29}$
- polovina maximálního rozsahu $H = 2 * Q_1 = 2^{30}$
- tři čtvrtiny maximálního rozsahu $Q_3 = 3 * Q_1 = 3 * 2^{29}$

Zvětšení intervalu

E1 – interval I je v dolní polovině zvoleného rozsahu čísla

- $I \subseteq \langle 0, H - 1 \rangle$
- na výstup se zašle hodnota 0
- interval I se nahradí novým intervalem $I = \langle 2 * low, 2 * high + 1 \rangle$

E2 - interval I je v horní polovině zvoleného rozsahu čísla

- $I \subseteq \langle M, H \rangle$
- na výstup se zašle hodnota 1
- nový interval $I = \langle 2 * (low - H), 2 * (high - H) + 1 \rangle$

E3 - interval I je ve střední oblasti zvoleného rozsahu čísla

- $I \subseteq \langle Q_1, Q_3 - 1 \rangle$
- nový interval $I = \langle 2 * (low - H), 2 * (high - H) + 1 \rangle$
- bezprostřední kódování změny na výstup již nelze provést → viz dále

Zvětšení intervalu E3

- zavedeme čítač případů E3
  - na počátku bude mít nulovou hodnotu
- při každém výskytu případu E3 zvýšíme čítač o hodnotu 1
- jakmile nastane případ E1 nebo E2
  - zakódujeme příslušnou hodnotu 0 nebo 1
  - za ní tolik opačných hodnot (1 nebo 0), kolik je v čítači E3 evidováno výskytů případu E3
  - čítač E3 pak vynulujeme

Algoritmus dekódování

1. Počáteční hodnoty kumulativních četností $g_0, \ldots, g_n$ a interval I jsou stejné jako při kódování. Hodnota aritmetického kódu se v průběhu dekódování používá ve stejné délce, jakou používá interval I, tj. jako 31-bitové číslo bez znaménka. Označme proměnnou s hodnotou kódu písmenem v. Na začátku do proměnné v vložíme prvních 31 bitů aritmetického kódu.
2. Dekódování znaku:
   - Musíme najít interval, který obsahuje hodnotu aritmetického kódu v. Tedy musíme najít takovou hodnotu indexu i, pro kterou je splněno:
     - $low + s * g_{i-1} \leq v \leq low + s * g_{i-1}$, kde $s = (high - low + 1)/g_n$
     - upravíme $low + s * g_{i-1} \leq v < low + s * g_i$
     - a dále $g_{i-1} \leq (v - low)/s < g_i$
     - zjevně stačí vypočítat kumulativní četnost c odpovídající hodnotě kódu v: c = (v - low)/s
     - a vyhledat, do kterého intervalu kumulativních četností <gi-1,gi) hodnota c patří $g_{i-1} \leq c < g_i$

Princip

Jednoduchá varianta aritmetického kódování

- místo po znacích znaků kóduje text po bitech
- vstupem jsou číslice 0 a 1
- tyto dvě číslice rozděluje na
  - méně pravděpodobná číslice (MPS – méně pravděpodobný symbol)
    - pravděpodobnost výskytu p, kde $p \in (0, 0.5\rangle$
  - více pravděpodobná (VPS – více pravděpodobný symbol)
    - pravděpodobnost výskytu $1 - p$
- opět dochází ke zmenšováním intervalu I
  - na začátku je opět $I = \langle 0, 1)$
  - při kódování je ale zmenšován tak, že dolní mez zůstává stále 0
  - průběžná hodnota intervalu je $I = \langle 0, h)$

Princip

Kódování VPS

- interval $\langle 0, h)$ nahrazen podintervalem $\langle 0, (1-p) * h)$

Kódování MPS

- od mezí podintervalu $\langle (1-p) * h, h)$ je odečtena hodnota $(1-p) * h$

Hodnota aritmetického kódu c se počítá průběžně během kódování

- Na začátku je kód roven nule
- Při kódování VPS se jeho hodnota nemění.
- Při kódování MPS se k ní přičítá hodnota $(1-p) * h$

Pravidla kódování

- VPS: $c \rightarrow$ nezměněn, $\langle 0, h) \rightarrow \langle 0, (1-p) * h)$
- MPS: $c \rightarrow c + (1-p) * h$, $\langle 0, h) \rightarrow \langle 0, p * h)$

Algoritmus kódování

1. Začátek

- hodnotu intervalu I položíme $I = \langle 0, 1)$
- počáteční hodnotu kódu položíme $c = 0$.
- stanovíme, který symbol MPS a určíme jeho pravděpodobnost p

2. Průběžný krok - kódování jednotlivých binárních číslic

- Ze vstupu v každém kroku odebereme binární číslici
- Je-li tato VPS
  - vypočítáme novou hodnotu horní meze intervalu $h = (1 - p) * h$
- Je-li tato MPS
  - vypočítáme novou hodnotu kódu $c = c + (1 - p) * h$
  - určíme novou hodnotu horní meze intervalu $h = p * h$

Krok 2 opakujeme do vyčerpání vstupního textu

Příklad kódování

- Budeme kódovat binární číslo `01100`
- MPS je v tomto případě číslice `1`, její pravděpodobnost výskytu v čísle je $p = 0.4$

Vstupní bit	Hodnota kódu c	Horní mez intervalu h
0	0	$0.6 * 1 = 0.6$
1	$0 + 0.6 * 0.6 = 0.36$	$0.4 * 0.6 = 0.24$
1	$0.36 + 0.6 * 0.24 = 0.504$	$0.4 * 0.24 = 0.096$
0	0.504	$0.6 * 0.096 = 0.0576$
0	0.504	$0.6 * 0.0576 = 0.03456$

Algoritmus dekódování

1. Na počátku

- hodnotu intervalu I položíme $I = \langle 0, 1 \rangle$
- Zadáme MPS a jeho pravděpodobnost p

2. Průběžný krok – dekódování jednotlivých binárních číslic

- Srovnáme hodnoty c a $(1 - p) * h$
- Jestliže platí $c < (1 - p) * h$:
  - dekódujeme VPS
  - vypočítáme $h = (1 - p) * h$
- V opačném případě $c >= (1 - p) * h$:
  - dekódujeme MPS
  - vypočítáme $c = c - (1 - p) * h$, $h = p * h$

Příklad dekódování

Budeme dekódovat kód $c = 0.504$ z minulého příkladu

h	Kód c	$(1-p)\cdot h$	Dek	Nový kód c	Nová mez h
1	0.504	0.6	0		$0.6 \cdot 1$ 0.6
0.6	0.504	0.36	1	$0.504 - 0.36$ 0.144	$0.4 \cdot 0.6$ 0.24
0.24	0.144	0.144	1	$0.144 - 0.144$ 0	$0.4 \cdot 0.24$ $= 0.096$
0.096	0	0.0576	0		$0.6 \cdot 0.096$ 0.0576
0.0576	0	0.03456			$0.6 \cdot 0.0576$ 0.03456

Děkuji za pozornost!